## Remarks

The claimed invention concerns recursive querying that facilitates identifying dependencies for database code modules. Information concerning identified dependencies is, in some examples, stored in a dependency graph. None of the references cited by the Office Action disclose or suggest dealing with code objects, recursively querying a database catalog, and/or producing a dependency graph. Thus, applicant respectfully requests that the Examiner withdraw the rejections of claims 9-29 and allow the application.

## Specification Objection Under 35 U.S.C. §112

The Office Action asserts that the specification is "replete with terms which are not clear, concise and exact." The Office Action then provides one example of a typographic error on page 2, line 3, wherein a "5" was mistakenly typed as a "6". The specification has been amended to correct this typographic error. Applicant has reviewed the specification and has not found other examples of unclear, inexact and/or verbose terms.

## Claim Rejections Under 35 U.S.C. §112

The Office Action rejects claim 9 as containing a single step/means, relying on 35 U.S.C. §112, first paragraph, MPEP §2164.08(a), and *In re Hyatt*, 708 F. 2d 712 (Fed. Cir. 1983). Claim 9 is not written in means plus function form and thus *In re Hyatt* is inapplicable. The claim at issue in *In re Hyatt* was "a so-called 'single means claim,' that is, a claim drafted in 'means-plus-function' format yet reciting only a single element instead of a combination." *In re Hyatt*, p. 713. Similarly, MPEP §2164.08(a) reads "where a **means** recitation does not appear in combination...". (emphasis added). Since claim 9 is not a "means" claim, MPEP §2164.08(a) is inapplicable. Even if *In re Hyatt* and MPEP §2164.08(a) were applicable, they are incorrectly applied here since Claim 9 contains more than a single step. A first query is performed that queries a database catalog for direct dependencies of a code object. Then, for each dependency found, second queries are performed recursively until all the basic dependencies in the database catalog are found and a dependency tree is generated. Consequently, there are at least two steps, the first query and the subsequent recursive queries.

**Claim Rejections Under 35 U.S.C. §103**

MPEP §2142 reads:

> "To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves, or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, **the prior art reference** (or references when combined) **must teach or suggest all the claim limitations**. The teaching or suggestion to make the claim or a combination and reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure." (emphasis added) *In re Vaeck*, 947 F.2d 488 (Fed. Cir. 1991). The Office Action identifies no suggestion or motivation to combine the references provided, thus, the combination is improper.

MPEP §2143.03 reads "to establish *prima facie* obviousness of a claimed invention, all the claimed limitations must be taught or suggested by the prior art." *In re Royka,* 490 F.2d 981 (CCPA 1974). The references upon which the Office Action relies neither teach nor suggest all the claimed limitations of the invention, and thus the §103 rejections should be withdrawn. For example, none of the references, alone or in combination, teach or suggest processing DBMS code objects, recursively querying a database catalog and/or building a dependency graph that stores information concerning the dependencies between the code objects discovered in the database catalog.

MPEP §2143.03 reads "if an independent claim is not obvious under 35 U.S.C. §103, then any claim depending therefrom is not obvious." *In re Fine,* 837 F.2d 1071 (Fed. Cir. 1988). Thus, as illustrated below, since none of the independent claims are obvious, then neither are the claims that depend from the independent claims.

Claims 9, 17, 22, and 27-29

Claims 9, 17, 22, and 27-29 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Green (U.S. Patent No. 4,829,427).

Claim 9 reads:

> A method of generating a basic dependency tree of a code object that does not take into consideration dependencies on triggers and on

implementations of object oriented code objects, the method comprising the steps of:

querying a database catalog for direct dependencies of a code object and then for each dependency found, doing the query recursively until all basic dependencies are generated into a dependency tree.

While Green mentions databases, it does not teach or suggest database code objects, dependencies of code objects, recursively querying a database catalog, and/or generating a dependency tree. Accordingly, Green omits elements of claim 9 and thus does not render claim 9 obvious.

The Office Action asserts that "Green teaches code generators have the same dependencies as optimizers (col. 1, lines 32-67)". The use of the word "dependencies" in Green is unrelated to the "code object dependencies" at issue in the present application. To understand how the word "dependencies" is being used in Green, the Examiner's attention is directed to the following statement found in the background section of Green. "Because of the dependencies of the optimizer on the implementation of the data base system and the digital data processing system, any change in either has required extensive modifications of the optimizer." (col. 1, lines 59-63). Thus, Green is explaining that like an optimizer relies on a specific implementation of a database to work correctly and that changing the database requires changing the optimizer, so too would such changes require changing the code generator. Green uses "dependencies" in a non database term of art manner to mean "if there is a change in the implementation of database catalog 217, a change in the file system, or a change in the digital computer system, large portions of DML compiler 601 must be rewritten." (Col. 10, lines 22-25). Green is explaining that prior art code generators and optimizers share the same problem having been so closely tied to the database that when you change one you have to change the other. This is clearly not a "code object dependency" as the term is used in the application or in the database arts. Example dependencies, as employed in the application, are described with reference to Figure 8 on pages 14 and 15 in the application.

In analyzing Green, the Office Action references col. 6, lines 5-34. This section does not teach or suggest recursively querying a database catalog for code object dependencies and generating a dependency graph. Similarly, col. 7, lines 58-67 are silent concerning recursively

querying a database catalog for code object dependencies and generating a dependency graph as claimed in claim 9. The other sections cited by the Office Action; col. 8, lines 1-25, col. 9, lines 4-41, col. 14, lines 14-60, and col. 19, lines 21-30 are similarly devoid of teachings or suggestions concerning recursively querying a database catalog for code object dependencies and generating a dependency graph. Green is, quite simply, missing limitations claimed in the application and thus does not satisfy the MPEP §2143.03 standard for a §103 rejection.

Claim 17 reads:

> A method of generating dependency information including dependencies of code objects on database triggers, the method comprising the steps of:
>
> 1) using a recursive algorithm for querying a database catalog for direct dependencies of a code object and then for each dependency found, doing the query recursively until all basic dependencies are generated into a dependency graph;
>
> 2) using a parser on each of the code objects in the dependency graph to identify DML statements that "fires" triggers thereby identifying dependencies on triggers;
>
> 3) applying the recursive algorithm on each of the triggers to incorporate the dependencies of the triggers into the dependency graph; and
>
> 4) repeating steps 1-3 for incorporating dependencies on triggers and their dependencies until new dependencies are not added to the dependency graph.

The Office Action asserts that Green discloses the claimed limitations in claim 17. However, Green omits elements of claim 17. To wit, Green is devoid of teaching or suggestion concerning a recursive algorithm for querying a database catalog for dependencies of code objects and/or triggers, and building a dependency graph. In particular, none of the sections in Green referenced by the Office Action (e.g., col. 5, lines 25-26, col. 6, lines 1-45, col. 7, lines 58-67, col. 8, lines 1-25, col. 9, lines 4-41, col. 14, lines 14-60, and col. 19, lines 21-30) recite

even one of the listed claim limitations. Thus, since it is missing claimed elements, Green does not satisfy the MPEP §2143.03 standard for a §103 rejection of claim 17.

Claim 22 reads:

> A method of generating dependencies of code objects as well as implementations of object oriented code objects in a database, the method comprising the steps of:
>
> 1) applying a recursive algorithm that queries a database for dependency information and outputs a direct dependency graph of a database code object, the "direct dependency graph" containing dependencies that do not involve dependencies on triggers and on implementations of object-oriented code objects in the database;
>
> 2) applying the recursive algorithm on each of the object-oriented code objects in the dependency graph to incorporate dependencies of implementations of code objects in the database; and
>
> 3) repeating steps 1 and 2 for incorporating dependencies of implementation of object-oriented code objects until new dependencies are not added in the dependency graph.

The Office Action asserts that Green discloses the claimed limitations in claim 22. However, Green omits elements of claim 22. Specifically, Green does not teach or suggest a recursive algorithm for querying a database for dependencies of code objects, using the recursive algorithm to incorporate dependencies of implementations of code objects, and building a direct dependency graph. In particular, none of the sections referenced by the Office Action (e.g., col. 6, lines 5-34, col. 7, lines 58-67, col. 8, lines 1-25, col. 9, lines 4-41, col. 14, lines 11-60, and col. 19, lines 21-30) recite even one of the listed claimed limitations. Thus, since it is missing claimed elements, Green does not satisfy the standard for a §103 rejection of claim 22.

Claim 27 reads:

> A system for identifying dependencies, if any, of a target data base code object, the system comprizing:

a digital computer;

a database server coupled to the computer;

a database coupled to the database server having data stored therein, the data comprizing object-oriented code objects, specifications of packages, implementations of packages, specifications of types, implementations of types and triggers; and

a code mechanism for generating a dependency graph, the dependency graph being a data structure and having entries to contain representations of depending code objects, specifications of packages, implementations of packages, specifications of types, implementations of types, triggers and dependencies of triggers that are relevant to the target data base code object.

The Office Action asserts that Green discloses the claimed limitations in claim 27. However, Green omits elements of claim 27. Specifically, Green does not teach or suggest data comprising object-oriented code objects, specifications of packages, implementations of packages, specifications of types, implementations of types and triggers, and a code mechanism for generating a dependency graph as claimed in claim 27. In particular, none of the sections referenced by the Office Action (e.g., col. 1, lines 15-66, col. 3, lines 32-55, col. 6, lines 5-21, col. 6, lines 5-34, col. 7, lines 58-67, col. 8, lines 1-25, col. 9, lines 4-41, col. 14, lines 11-60, and col. 19, lines 21-30) recite even one of the claimed limitations mentioned above. Thus, since it is missing claimed elements, Green does not satisfy the standard for a §103 rejection of claim 27.

Claim 28 reads:

A method for generating dependencies of a target data base code object, using a computer having a processor, memory, display, input/output devices, the method comprising the steps of:

providing a data base coupled to the computer having data stored therein, the data comprizing representations of object-oriented code objects, specifications of packages, implementations of packages, specifications of types, implementations of types and triggers; and

using a recursive code mechanism for generating a dependency graph, the dependency graph being a data structure and having entries to contain representations of dependant code objects, specifications of packages, implementations of packages, specifications of types, implementations of types, triggers and dependencies of triggers which are relevant to the target data base code object.

The Office Action asserts that Green discloses the claimed limitations in claim 28. Green omits elements of claim 28. Green does not teach or suggest a database storing data representations of object-oriented code objects, specifications of packages, implementations of packages, specifications of types, implementations of types and triggers. Green also does not teach using a recursive code mechanism for generating a dependency graph as claimed. In particular, none of the sections referenced by the Office Action (e.g., col. 6, lines 5-34, col. 7, lines 58-67, col. 8, lines 1-25, col. 9, lines 4-41, col. 14, lines 11-60, and col. 19, lines 21-30) recite any of the claimed limitations mentioned above. Thus, since it is missing claimed elements, Green does not satisfy the standard for a §103 rejection of claim 28.

The Office Action also asserts that independent claim 29 is rejected under Green. However, the Office Action presents no arguments concerning the rejection of claim 29 under Green and thus this rejection should be withdrawn.

Claims 10-16, 18-21, and 23-26

Claims 10-16, 18-21, and 23-26 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Green in view of McKeeman (U.S. Patent No. 5,651,111).

Claim 10 depends from claim 9. Accordingly, claim 10 is not obvious and is allowable for the same reasons set forth regarding claim 9. Furthermore, neither Green nor McKeeman provide any motivation or suggestion to combine the references. Additionally, McKeeman is silent concerning database code objects, recursive algorithms for querying a database catalog to determine dependencies and building a dependency graph. The Office Action provides no rationale for why Green and McKeeman should be combined in the proposed manner. Thus, the

rejection does not meet the standard for a valid combination of references according to MPEP §2142.

Even if combined, the references still don't disclose the invention. McKeeman does not disclose the additional claim limitation of claim 10 of using a generated dependency tree to compile code objects in debug mode as part of a database code object debugging tool and thus does not render claim 10 obvious. McKeeman also does nothing to overcome the shortcomings of Green, similarly omitting claimed elements. Therefore, neither Green nor McKeeman, alone or in combination, teach or suggest the claim limitations in claim 10 as required by MPEP §2142.

Claims 11-16 depend from claim 9. Since claim 9 is not obvious, neither are claims 11-16. Furthermore, the rejections of claims 11-16 do not meet the standard for a valid combination of references.

Additionally, McKeeman does not disclose the additional claim limitation of claim 11 of employing a dependency tree to identify calling paths in a database code coverage tool. While McKeeman mentions code coverage tools in general, there is no discussion of a code object dependency tree in either Green or McKeeman, and no use of a database code coverage tool that employs the dependency tree. Furthermore, McKeeman does not disclose the additional claim limitation of claim 12 of using the generated dependency tree to identify calling paths in a database code object profiling tool. McKeeman also does not disclose the additional claim limitation of claim 13 of using the generated dependency tree in a database code object testing tool. McKeeman also does not disclose the additional claim limitation of claim 14 of using the generated dependency tree to identify dependent objects that are invalid in the database. While McKeeman mentions testing input values in a database to see whether they are invalid, this is unrelated to analyzing invalid object dependencies. McKeeman also does not disclose the additional claim limitation of claim 15 of using the generated dependency tree to identify cyclic dependencies among database code objects. McKeeman also does not disclose the additional claim limitation of claim 16 of using a generated dependency tree in a dependency graph presentation tool. Therefore, neither Green, nor McKeeman, alone or in combination, teach or suggest the claim limitations in claims 12, 13, 14, 15, or 16 as required by MPEP §2142.

Claims 18-21 depend from claim 17. Since claim 17 is not obvious, neither are claims 18-21. The rejections of claims 18-21 do not meet the MPEP §2142 standard for a valid combination of references.

McKeeman does not disclose the additional claim limitation of claim 18 of employing a dependency tree to identify calling paths in a database code coverage tool. Similarly McKeeman does not disclose the additional claim limitation of claim 19 of using the generated dependency tree to identify calling paths in a database code object profiling tool. McKeeman also does not disclose the additional claim limitation of claim 20 of using the generated dependency tree in a database code object testing tool. McKeeman also does not disclose the additional claim limitation of claim 21 of using the generated dependency tree to identify dependent objects that are invalid in the database. Therefore, neither Green, nor McKeeman, alone or in combination, teach or suggest the claim limitations in claims 18, 19, 20, or 21 as required by MPEP §2142.

Claims 23-26 depend from claim 22. Green does not disclose the elements claimed in claim 22 and does not render it obvious. Thus, since claim 22 is not obvious, neither are claims 23-26. Furthermore, the rejections of claims 23-26 do not meet the MPEP §2142 standard for a valid combination of references.

McKeeman does not disclose the additional claim limitation of claim 23 of employing a dependency tree to identify calling paths in a database code coverage tool. Similarly McKeeman does not disclose the additional claim limitation of claim 24 of using the generated dependency tree to identify calling paths in a database code object profiling tool. McKeeman also does not disclose the additional claim limitation of claim 25 of using the generated dependency tree in a database code object testing tool. McKeeman also does not disclose the additional claim limitation of claim 26 of using the generated dependency tree to identify dependent objects that are invalid in the database. Therefore, neither Green, nor McKeeman, alone or in combination, teach or suggest the claim limitations in claims 23, 24, 25, or 26 as required by MPEP §2142.

Claim 29

Claim 29 stands rejected under 35 U.S.C. §103(a) as being unpatentable over Green in view of McKeeman (U.S. Patent No. 5,651,111). Claim 29 is an independent claim that reads:

> A computer program product embedded on a computer readable medium for use in debugging a target data base code object comprizing:

a recursive code mechanism for generating a dependency graph of the target data base code object, the dependency graph being a data structure and having entries to contain representations of depending code objects, specifications of packages, implementations of packages, triggers and dependencies of triggers which are relevant to the target data base code object; and

a program code mechanism for using the dependency graph to debug the target data base code object.

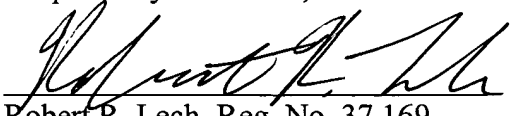
Neither Green nor McKeeman disclose claimed elements of claim 29. For example, neither Green nor McKeeman, alone or in combination, teach or suggest a recursive code mechanism for generating a dependency graph, a dependency graph data structure with entries for representations of depending code objects, specifications of packages, implementations of packages, triggers and dependencies of triggers, and/or a program code mechanism for using the dependency graph to debug the target data base code object.

Additionally, it is to be noted that claim 29 claims a computer readable medium. Neither Green nor McKeeman disclose a computer readable medium on which a system and/or method for use in debugging a target database code object is stored. Thus, for claim 29, the combination of Green and McKeeman does not satisfy the standard for a §103 rejection as listed in MPEP §2143.03.

## Conclusion

Based on the foregoing remarks and amendments, the applicant believes that all of the claims in this case are in a condition for allowance and an indication to that effect is earnestly solicited. Furthermore, if the Examiner believes that additional discussions or information might advance the prosecution of this case, the Examiner should feel free to contact the undersigned at the telephone number indicated below.

Respectfully submitted,

By: 

Robert R. Lech, Reg. No. 37,169
Calfee, Halter & Griswold LLP
1650 Fifth Third Center
21 East State Street
Columbus, Ohio 43215-4243
Phone (614) 621-7101
Fax (614) 621-0010

Markup of Original first Paragraph on page 2:

Yet another attempt[,] involved generating complementary source code to resolve external dependencies of program units to facilitate unit testing, as described in US Patent No. [6]5,651,111 titled "Method and Apparatus for producing a software test system using complementary code to resolve external dependencies." Yet another attempt[,] involved building an object oriented software program using compiler generated direct dependency information described in US Patent No. 5,758,160 titled "Method and apparatus for building a software program using dependencies derived from software component interfaces". In all of these cases only the direct dependencies of the modules concerned are used. In a database programming environment, spending an inordinate amount of time finding the dependencies of program units using a compiler is undesirable, since that information is directly available from the database catalog. Many other US patents describe various debugging and testing systems but none of these which is known to Applicant provides the method and system of the present invention for automatically generating the complete dependencies necessary to debug code objects.